PorePy: Simulation software for mixed-dimensional problems

Eirik Keilegavlen

INSPIRE Summer School – Geilo August 2019

Content

- Part I: Introduction to mixed-dimensional problems
- Part II: Technical aspects (software design, PorePy specifics)
- Part III: Multiphysics simulations – jupyter notebooks

Part I: Mixed-dimensional problems (in PorePy)

Outline of Part I

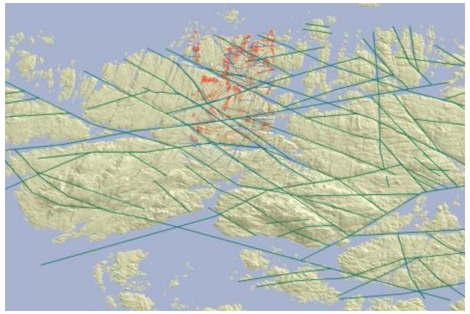
- Motivation: Examples of mixed-dimensional problems
- Challenges to software design
- Example: Mixed-dimensional flow problem
    - Modeling
    - Discretization
    - PorePy implementation and usage

Examples of mixed-dimensional problems / geometries

Geometry

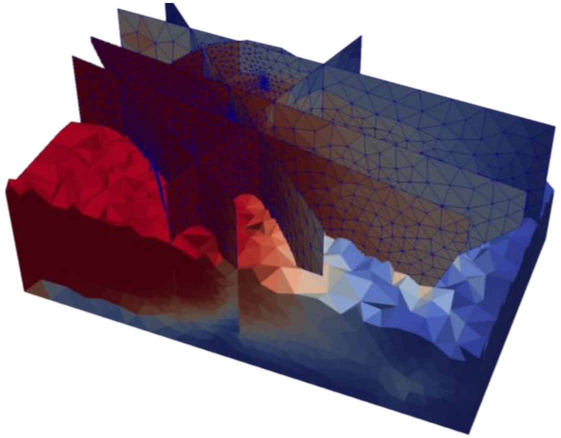
Centimeters

Meters

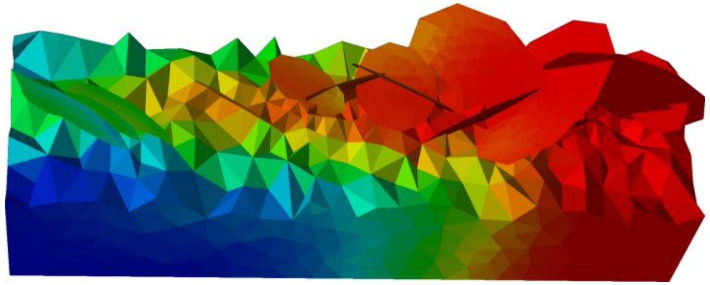


Kilometers

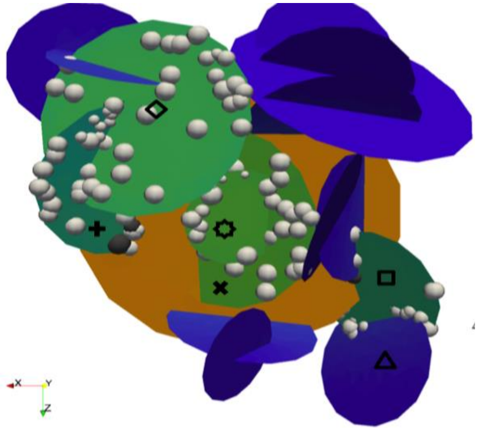
Processes



Flow



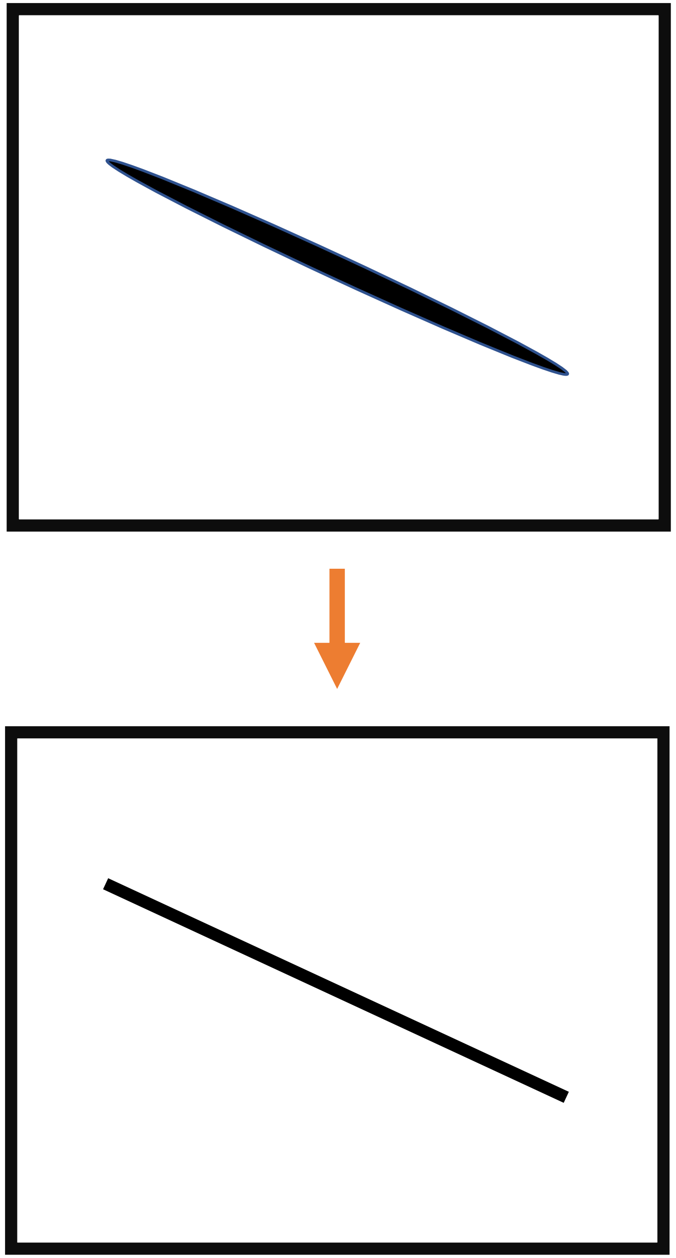
Transport



Deformation

Mixed-dimensional problems

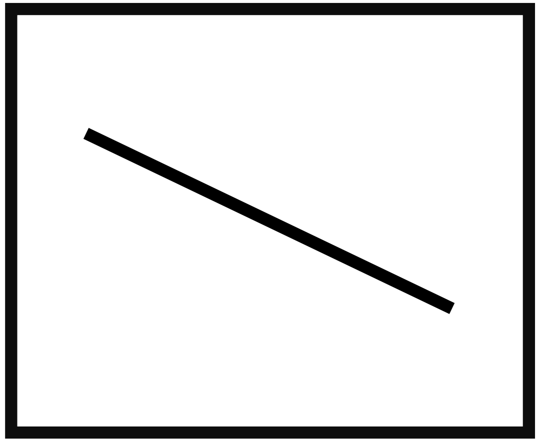
- Starting point: The simulation domain contains inclusion with high aspect ratio
- Governing equations posed both in inclusion and surroundings
- Lower-dimensional problem obtained by averaging
  - Inclusion represented as embedded manifold
- Mixed-dimensional problem formed by
- Herein: Only dimension gaps of 1

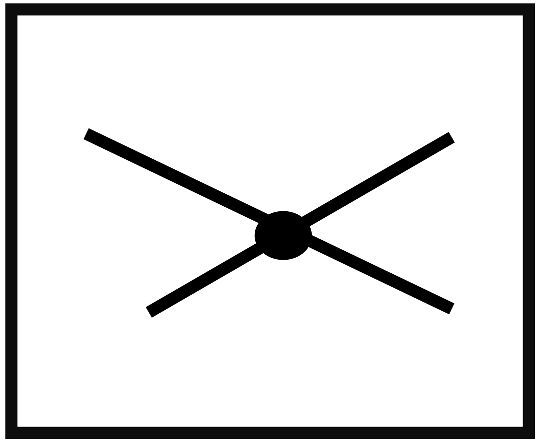


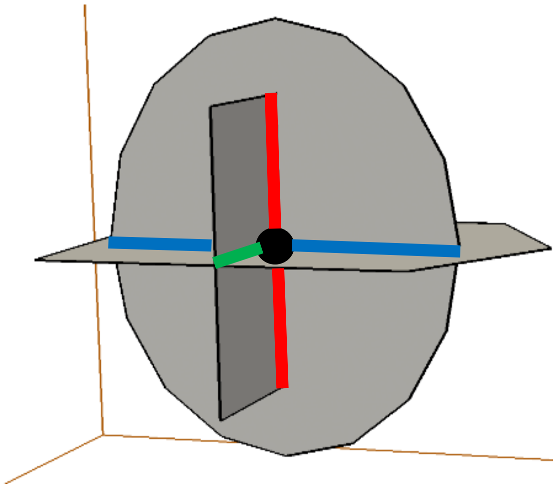
‘Fractures’ can include anything long and thin that may be treated with dimension reduction. Subsurface examples:

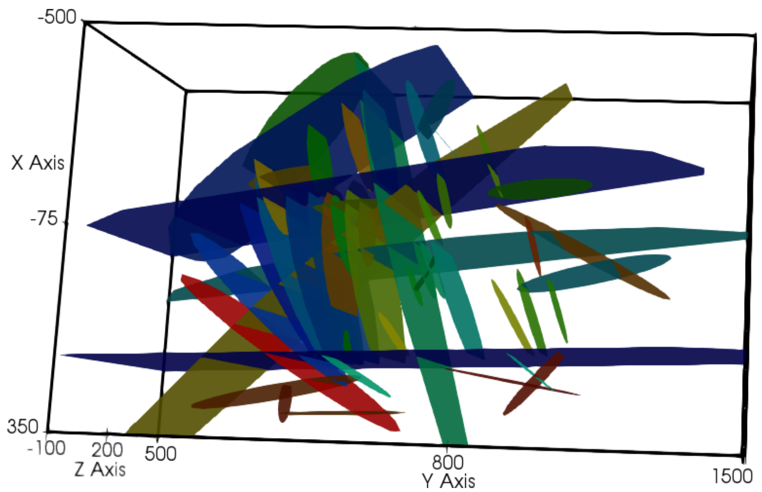
- Real fractures
- Near-fault regions
- Aquifers (Carbon storage)

Modeling in complex geometries









(Coupled) processes in mixed-dimensions

- Flow in fracture network and the host medium
- Transport (advection-diffusion)
- Chemical deposition within fractures
- Deformation of the host medium
- Dynamic fractures:
  - Opening of existing fractures
  - Sliding of existing fractures (induced seismicity / earthquakes)
  - Fracture propagation

Representation in standard simulation tools requires the processes be upscaled (represented by averaging).

Depending on the application, this can be okay, or an illusion.

PorePy:

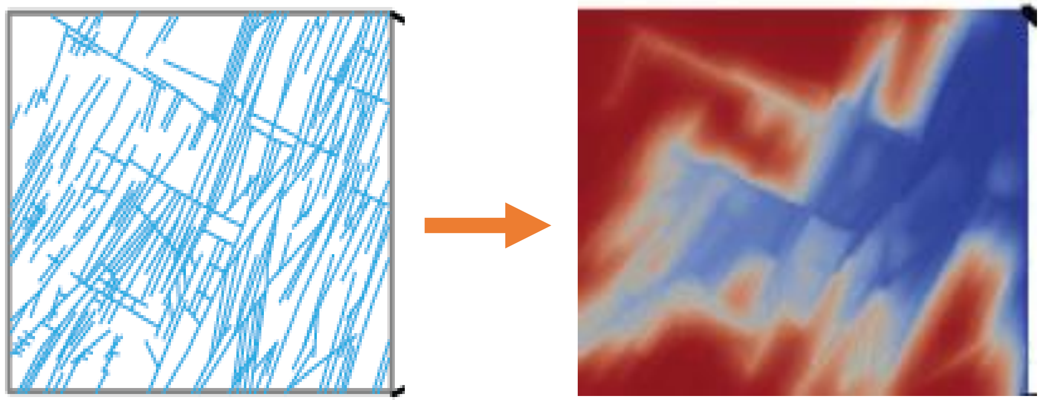
- Python framework for dynamics in fractured porous media
- Build as a multi-physics framework for mixed-dimensional geometries
- Usage: Test numerical methods, models, application-motivated simulations
- Emphasis on rapid prototyping
  - Computational efficiency prioritized within constraints of the framework

- Development team: 5-6 people, mainly PhD students
- Development started in 2017 (2016), open sourced May 2017

www.github.com/pmgbergen/porepy

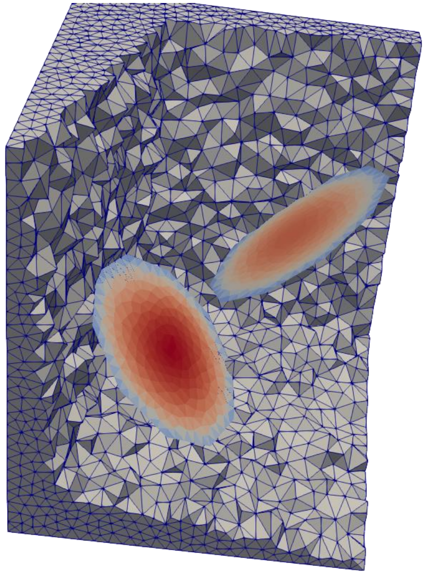
Functionality

- Meshing of domains with complex internal constraints
- Discretization schemes for flow, transport, poro-mechanics
- Strong emphasis on coupling between physics and dimensions



Flow and transport

Video: Unstable displacement.



Fracture deformation

Model problem: Mixed-dimensional elliptic equation

- Goal: Define a mixed-dimensional version of the elliptic model problem.
- Solve with PorePy
- Ingredients:
  - Geometry
  - Model equations
  - Discretization strategy, emphasis on reuse of existing implementation
  - Implementation

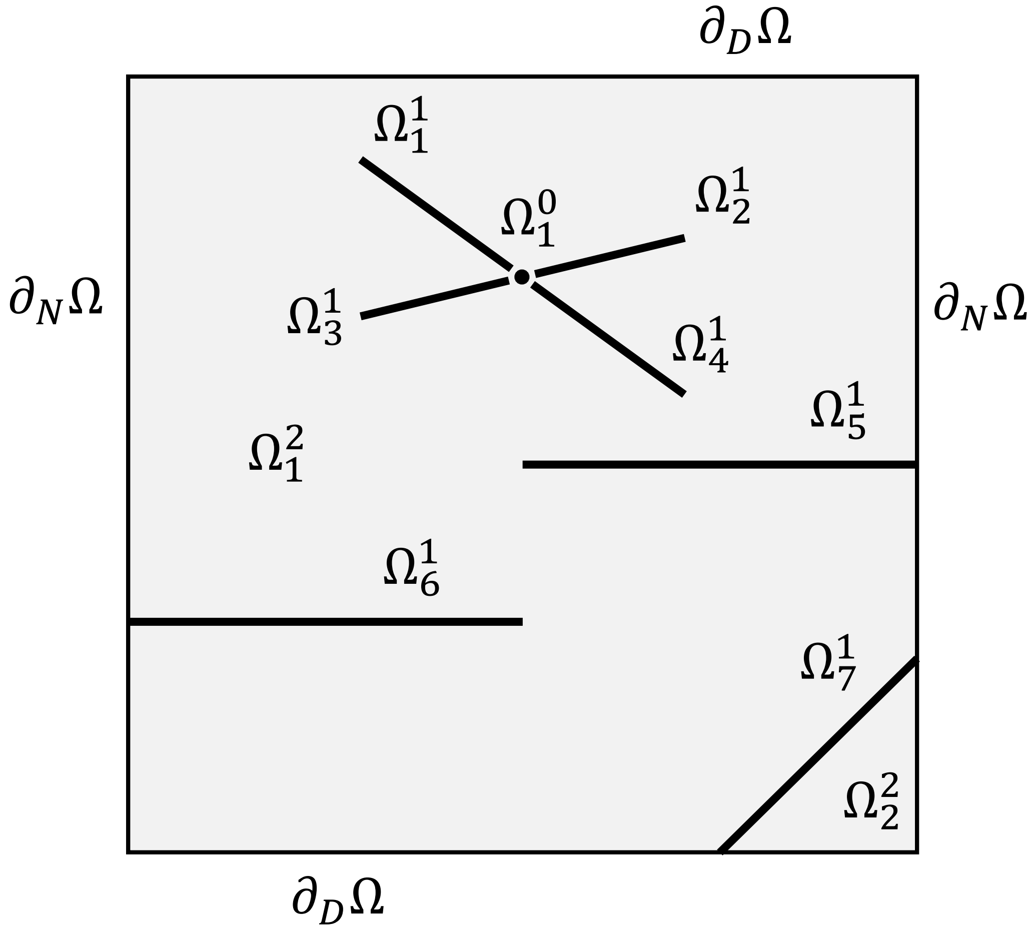
Geometry I: Subdomains

Partition into subdomains

- Ω_i^D: Matrix
- Ω_i^{D-1}: Fractures
- Ω_i^{D-2}, (Ω_i^{D-3}): Fracture intersections

Also need to deal with boundary between subdomains

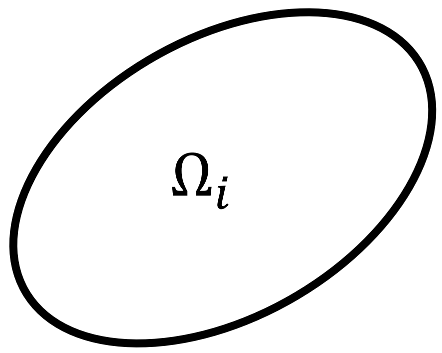
Next slides: Occasionally drop dimension superscript



Geometry II: Notation

- Ω_i: Generic subdomain (matrix, fracture, intersection)
- Γ_j: Generic interface between subdomains

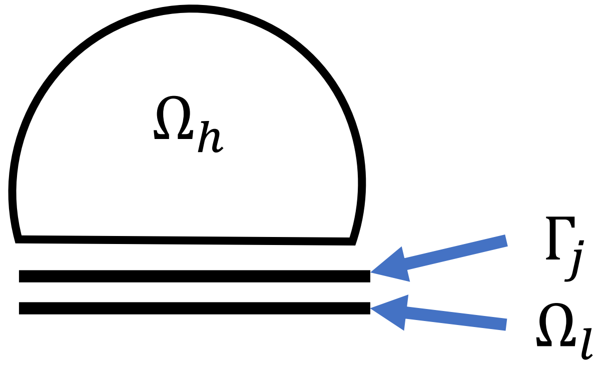
Generic variables marked by subscripts: p_i, λ_j



Interaction between subdomains

- Ω_l: Lower-dimensional neighbor
- Γ_j: Interface
- Ω_h: Lower-dimensional neighbor

Geometrically, Ω_l, Γ_j, $\partial_j\Omega_i$ coincide.

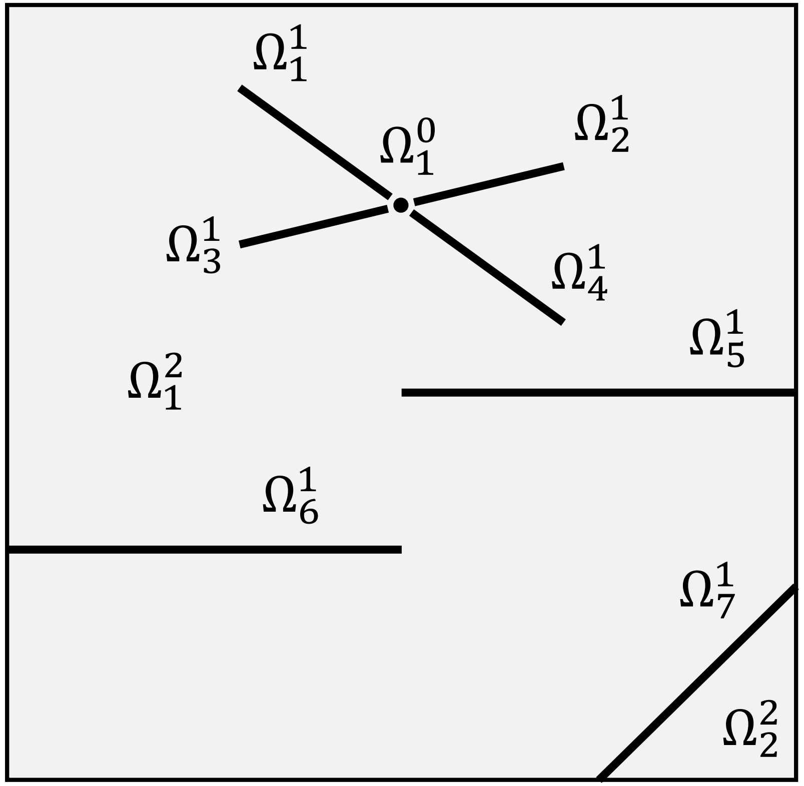


Note: No + and - side

Geometry III: Neighbors

For a subdomain Ω_i:

- $\widehat{S}_i$ is the set of interfaces to higher-dimensional subdomains
- $\check{S}_i$ is the set of interfaces to lower-dimensional subdomains

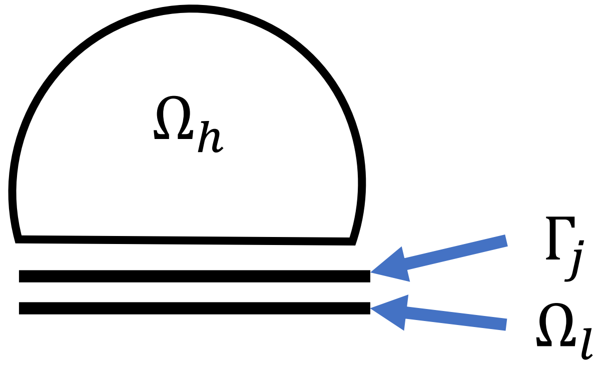


Geometry IV: Projections

- Ξ_i^j projects fluxes from Ω_i to Γ_j
- Ξ_j^i projects fluxes from Γ_j to Ω_i
- Π_i^j projects pressures from Ω_i to Γ_j
- Π_j^i projects pressures from Γ_j to Ω_i

Projections from Γ_j to Ω_h really goes to $\partial_j \Omega_h$

Extensive and intensive quantities require different projections



Governing equations for mixed-dimensional flow – strong form

Within subdomain Ω_i

Darcy's law:

$$\boldsymbol{u}_i + K_i \nabla p_i = \boldsymbol{0}$$

Conservation of mass:

$$\nabla \cdot \boldsymbol{u}_i = f_l + \sum_{j \in \widehat{S}_i} \Xi_j^i \lambda_j$$

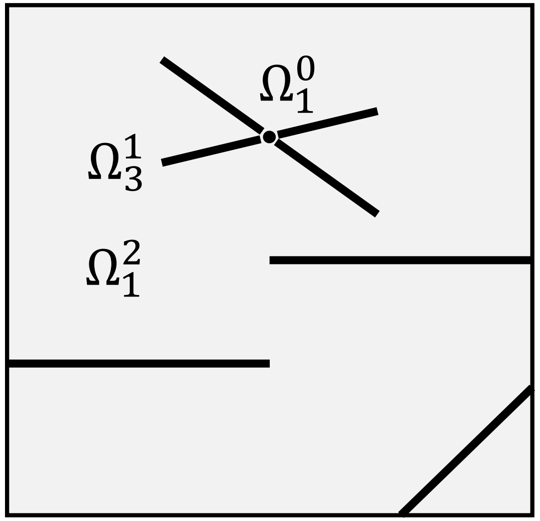
Boundary condition:

$$\boldsymbol{u}_i \cdot \boldsymbol{n}_j = \Xi_j^i \lambda_j, \qquad j \in \check{S}_j$$

Over interface Γ_j

Flux-pressure relation

$$\lambda_j + \kappa_j \left(\Pi_l^j p_l - \Pi_h^j tr\, p_h \right) = 0$$



Mathematical framework

The modeling emphasizes conservation:

- Integral formulation of the equations follows immediately
- Dual variational formulation uses function spaces

$$p_i \in L^2(\Omega_i), \qquad q_i \quad \in H(\nabla \cdot, \Omega_i),$$
$$tr(p_i) \in L^2\left(\partial_j \Omega_i\right), \qquad \lambda_j \in L^2\left(\Gamma_j\right)$$

- Exception: Contact mechanics, cast in primal form
- The modeling framework presented herein can be analyzed as a mixed-dimensional de Rahm complex

Discretization

- Subdomain problems have the same form as a fixed-dimensional pressure equation.
- Boundary conditions and source terms are not known.
- Formulate discretization centered on the interface.
- Subdomain discretization considered a black box that converts fluxes and sources to pressures and pressure traces

Darcy's law:

$$\boldsymbol{u}_i + K_i \nabla p_i = \boldsymbol{0}$$

Conservation of mass:

$$\nabla \cdot \boldsymbol{u}_i = f_l + \sum_{j \in \hat{S}_i} \Xi_j^i \lambda_j$$

Boundary condition:

$$\boldsymbol{u}_i \cdot \boldsymbol{n}_j = \Xi_j^i \lambda_j, \qquad j \in \check{S}_j$$

Interface:

$$\lambda_j + \kappa_j \left(\Pi_l^j p_l - \Pi_h^j tr\, p_h \right) = 0$$

```
class EllipticDiscretization():

    def discretize_neumann_flux(…):
        # Neumann boundary term from
        # lower-dimensional interface

    def impose_source_term(…):
        # Source term from
        # higher-dimensional interface

    def pressure_trace_discretization(…):
        # Pressure trace on
        # lower-dimensional interface

    def pressure_cell_discretization(…):
        # Pressure values for
        # higher-dimensional interfaces

    def discretize_standard_problem(…):
        # Standard existing code
```

All methods return discretization matrices
Actual function names are different (longer)

Darcy's law:

$$\boldsymbol{u}_i + K_i \nabla p_i = \boldsymbol{0}$$

Conservation of mass:

$$\nabla \cdot \boldsymbol{u}_i - \sum_{j \in \hat{S}_i} \Xi_j^i \lambda_j = f_i$$

Boundary condition:

$$\boldsymbol{u}_i \cdot \boldsymbol{n}_j = \Xi_j^i \lambda_j, \qquad j \in \check{S}_j$$

Interface:

$$\lambda_j + \kappa_j \left(\Pi_l^j p_l - \Pi_h^j tr\, p_h \right) = 0$$

Subdomain coupling via the interfaces

```
class RobinCoupling():

    def dicretize(discr_h, discr_l, …):

        # discretize the interface variable

        discr_h.discretize_neumann_flux(…)

        discr_l.impose_source_term(…):

        discr_h.pressure_trace_discretization(…):

        discr_l.pressure_cell_discretization(…):
```

$$\begin{pmatrix} 0 & 0 & A_{HM} \\ 0 & 0 & A_{LM} \\ A_{MH} & A_{ML} & A_{MM} \end{pmatrix} \begin{pmatrix} p_H \\ p_L \\ \lambda_M \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Discretization – pseudo code

1. Loop over all subdomains, apply discretization scheme
2. Loop over all interfaces
   1. Discretize the interface variable
   2. Impose on neighboring subdomains

$$\begin{pmatrix} A_1 & 0 & 0 & \ddots & \cdots & \ddots \\ 0 & \ddots & 0 & \vdots & S_{ij} & \vdots \\ 0 & 0 & A_N & \ddots & \cdots & \ddots \\ \ddots & \cdots & \ddots & B_1 & 0 & 0 \\ \vdots & T_{ji} & \vdots & 0 & \ddots & 0 \\ \ddots & \cdots & \ddots & 0 & 0 & B_M \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_N \\ \lambda_1 \\ \\ \lambda_M \end{pmatrix} = 0$$

External boundaries and source terms are ignored

No requirements on the subdomain discretizations, grids etc.

Discretization in PorePy: Steps

1. Define fracture network, create mesh
2. Set parameters
3. Define variables and discretization schemes
4. Apply discretization method
5. Assemble and solve linear system
6. Visualize, post process

.. over to jupyter